LogiTrace
INVENTORY MANAGEMENT

**Patrones de diseño:**

- **Tomás Nicolás Jerez Garcia.**
- **Jacobo Ulises Cortes Duque.**
- **Santiago Zamora Garzón.**
- **Gabriel Mateo Gonzalez Lara.**

Utilizar e implementar el patrón Singleton dentro del proyecto para la BD.

**Ques es y para que sirve en el proyecto?**

El patrón Singleton garantiza que solo exista una instancia de una clase y que sea accesible de forma global.
Se usa cuando una sola entidad debe controlar o coordinar operaciones en el sistema, como:

- Conexiones a base de datos
- Configuración global
- Control de inventario o stock

En un sistema de inventario, el Singleton se usaría para asegurar que todas las partes del programa (por ejemplo, los módulos de ventas, compras y almacén) trabajen con el mismo estado del inventario.

Así se evita que existan dos copias del stock (por ejemplo, una diga que hay 10 unidades y otra 8).

## 1. Singleton para la conexión a la base de datos

**Contexto:**

Django expone el ORM, pero los servicios de dominio necesitaban ejecutar SQL de diagnóstico sin duplicar conexiones ni credenciales.

**Solución:**

infraestructura/database.py define DatabaseConnection, un *Singleton thread-safe* que comparte connections['default'] en todo el *stack*.

**Implementación:**

__new__ controla la *instancia única* y _wrapper() inicializa una sola vez el *handler* de Django bajo demanda. execute() devuelve filas como diccionarios para reutilizarlas sin *boilerplate*.

LogiTrace
INVENTORY MANAGEMENT

**Uso:**

domain/services/database_health.py consume el *Singleton* para medir latencia y core/views.database_health expone /health/db/ en logitrace/urls.py.

**Beneficios:**

(1) un único punto de configuración, (2) menos conexiones abiertas, (3) facilidad para instrumentar métricas y auditoría del *driver*.

**Riesgos:**

si la conexión cae, DatabaseConnection cierra y reintenta en la siguiente solicitud. Recomendado monitorear el *endpoint* desde DevOps.

**Extensiones:**

el *Singleton* puede exponer *helpers* para transacciones largas o inyectarse en futuros repositorios de dominio.

elegir otro ( Factory), explicar un escenario de uso aunque no se implemente

.

**Porque elegimos este patron de diseño?**

Se utiliza el patrón de diseño Factory Method en el proyecto porque el sistema de gestión de stock debe manejar diferentes tipos de productos (por ejemplo, libros,juegos, componentes), cada uno con características y comportamientos distintos.

El patrón Factory permite centralizar y estandarizar la creación de estos productos, evitando que el código principal del sistema tenga múltiples condicionales o deba modificarse cada vez que se agregue un nuevo tipo de producto.

De esta manera, si en el futuro se requiere incorporar un nuevo tipo de producto (como “armas” o “peliculas”), solo será necesario crear una nueva clase y agregar su método de creación en la fábrica, sin alterar la lógica esencial del sistema ni comprometer su estabilidad.

Esto aporta flexibilidad, escalabilidad y bajo acoplamiento, principios fundamentales en la ingeniería de software para mantener un proyecto fácil de extender y mantener.

**Implementación:**

**Comando para que funcione :**

```
C:\Users\atomi>curl -X POST http://127.0.0.1:8000/products/factory/ -H "Content-Type: application/json" --data-raw "{\"s
ku\":\"SKU-100\",\"name\":\"Vacuna\",\"category\":\"perishable\",\"metadata\":{\"temperature\":\"4C\"},\"reorder_point\"
:3}"
{"sku": "SKU-100", "name": "Vacuna", "category": "perishable", "storage_instructions": "Mantener refrigerado entre 4C y
registrar fecha de caducidad.", "recommended_reorder_point": 15, "compliance_tags": ["cold-chain", "expiry-tracking"]}
C:\Users\atomi>
```

LogiTrace
INVENTORY MANAGEMENT

**Visualizacion de la implementación:**


```
engine:       "django.db.backends.mysql"
database:     "logitrace"
host:         "127.0.0.1"
port:         "3306"
checked_at:   "2025-10-25T03:46:50.716618+00:00"
status:       "online"
latency_ms:   2.979
```